Formal Methods in Industry

- Several UK and other European companies use Level 1 or 2 formal methods to develop requirements specifications
- GEC Alsthom (in France) use their expertise in formal methods as a major element in developing their business in high-speed train and conventional power station control
  ($18 billion backlog)
- With NASA encouragement (and some sponsorship) all the major US manufacturers of commercial avionics have active formal methods programs
  (Allied Signal, Boeing, Collins, Honeywell, IBM/Loral)

Status and Prospects

- Europe has commanding lead in PR concerning formal methods and tech-transfer (but the tech is low)
- North America has a commanding lead in tools to support formal methods
- Next big step will be integration of classical theorem-proving methods with state-exploration and other automatic techniques
- And integration with testing, prototyping etc.
- Need to better figure out the connections between formal methods and semi-formal techniques in software engineering such as object-oriented design
- Can get cost-effective return on formal methods *now* with a few highly skilled people working on the most difficult aspects of design

Conclusion

- Use formal methods to *augment*, not replace traditional methods
- Use where traditional methods are absent, ineffective, expensive
- Typically, requirements, and the hardest and trickiest aspects of design; early lifecycle in either case
- You need mechanized assistance to debug specifications
  - Need strong checks on consistency
  - And must probe consequences in order to validate spec'ns
- Don't worry that you won't have a complete formal "proof of correctness" from top to bottom—there's more than this to assurance for critical systems
- Use proofs as "instruments of discovery," and to guarantee absence of certain conceptual errors

To Learn More

- Browse papers and technical reports in `/pub/reports` on `ftp.cs.sri.com` (start with `readme.txt`)
- You can get our verification system PVS by anonymous FTP from `ftp.csl.sri.com` in `/pub/pvs`
  (Allegro Lisp implementation for Sun SparcStations; Recommend 32M real memory, 100M swap space, Sparc 2 or better)

Lifecycle Phases

Pros and cons of applying formal methods in early and late lifecycle phases

- Late lifecycle

  Pro: That's what runs

  Con: Size of description is large; must often leave purely functional world of ordinary logic (i.e., need VCGs, Hoare sentences); *traditional methods are very effective*

- Early lifecycle

  Pro: That's where the serious errors are; that's where the concern is; few other rigorous techniques available

  Con: front-loads development time and cost

Formal Methods and Critical Systems

- Intellectual argument that fully formal methods guarantee certain *modeled* properties to very high degree
- *Validity* depends on fidelity of the modeling employed
  - Surely easier to validate abstract models that make a few broad assumptions than detailed models with intricate assumptions
  - Most likely to arise early in lifecycle and at the highest levels of abstraction
- *Utility* must be evaluated against alternatives
  - Current techniques seem adequate at later stages of implementation
  - Mishaps are generally due to design flaws, multiple failures, unanticipated interactions, not coding errors

Formal Methods in the Past

Often applied to relatively routine aspects of system development at later lifecycle phases

- Verification of sequential program code
  - Very expensive (SACEM: 315,000 hours for 9,000 SLOC)
  - Adequately handled by conventional methods in safety/mission critical industries
- Flow analysis for TCB interface of trusted systems
  - Not considered main design or V&V issue in trusted systems
  - Often unconnected to the main development
- Formal verification of microprocessor datapaths
  - Not considered main processor design or V&V issue
  - Not taken up by industry

Making Effective Use of Formal Methods

- Apply where other methods are absent, ineffective, expensive
- And where it makes a difference (e.g., where faults could be catastrophic, or misunderstandings expensive to correct)
- Often the *hardest* and *trickiest* parts of design (fault-tolerance, timing, interrupt and MMU management in secure micro-kernels, coordination of parallel and distributed activities, control-dominated aspects of microprocessors)
- Best treated early in lifecycle in fairly abstract form, so validation is credible
- By relatively few, but highly skilled people
- Who are supported by effective tools
- To explore and expose the problem, and/or to guarantee that certain conceptual errors are not present

Partial Application of Formal Methods

- Cannot achieve total coverage; there will always be gaps that must be covered informally
- Must choose where those gaps should be, and how “big” they should be
- There are four axes of selectivity:
  - Degree of rigor
  - Components
  - Properties
  - Lifecycle phases

Degrees of Rigor in Formal Methods

- Can apply formal methods at many levels of rigor

  0. No use of formal methods
  1. Use the *ideas* of formal methods, but ad-hoc notation, proofs based on informal argument, tools are pencil and eraser (the way conventional mathematics is done)
  2. Formalize and maybe mechanize specification language and methodology, retain pencil and eraser for proofs
  3. Full mechanization with automated theorem proving or checking

- Roughly speaking, Europeans have focussed on Level 1 ($\leadsto$ 2), US on Level 3
- Higher levels are not necessarily better, but...

Formal Methods and Tools

- Formal specifications are no more likely to be correct than are programs (may not realize this unless you try proving theorems)
  - Need strong checks on consistency
  - And must validate specifications by probing their consequences

  Need mechanization to do this effectively
- Without verification, formal specifications are just documentation
- To contribute to assurance, you need to prove theorems

Selected System Components and Properties

- Most disciplines concerned with critical systems assign criticality levels to software components based on hazard analysis and potential consequences of failure (must consider malfunction and unintended function as well as loss of function)
  - Higher criticality levels seem good place to focus use of formal methods
- May not need to verify all functional properties
  - May be enough to guarantee a “safe” action, not necessarily an optimal one
  - In other cases, may want to verify properties other than function (e.g., security, or fault containment)
  - Often, absence of specific malfunctions is most important property

Assurance for Critical Systems

- Dependable systems: reliance may *justifiably* be placed in quality of service delivered
- Critical systems require very small failure rates (e.g., 10^{-9} per hour)
  This is the "ultra-dependable" region
- Require some *evidence* that this has been achieved
- Direct measurement requires 114,000 years on test
- Essentially all assurance has to derive from subjective factors concerning development processes
- In other words, in the ultra-dependable region, cannot provide evidence of *"how well you've done"* (i.e., dependability achieved), instead provide evidence of *"how hard you tried"*

Claims for Formal Methods

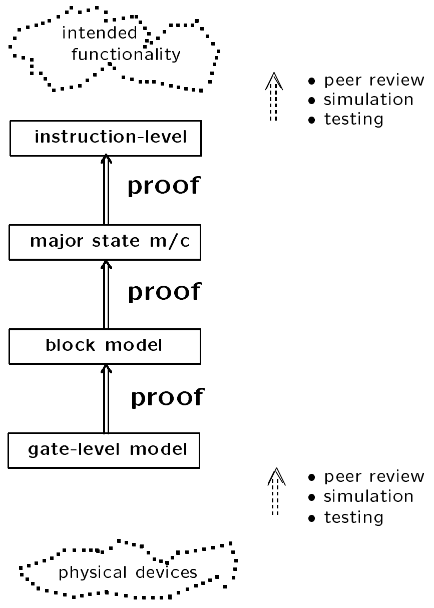
1. Enable faults (of assumptions, requirements, design) to be detected *earlier* than otherwise
   - Due to greater precision and explicitness early in the lifecycle
2. Enable faults to be detected with greater *certainty* than otherwise
   - Because they replace reviews (consensus) by analyses (calculations)
   - And can provide total coverage (of selected, modeled properties)
3. *Guarantee* absence of specified faults (subject to accuracy of modeling employed)
   - Because the calculations (proofs) can be checked mechanically (by a theorem prover)

Limitations of Formal Methods

- Formal methods are a modeling activity: they deal with mathematical *models* of reality
- Just like applied math in all other engineering disciplines
- To be useful, we must *validate* that our assumptions are true statements about the real world

  And that our formalized requirements accurately capture the real requirements
- These validation activities are necessarily informal, empirical and imperfect

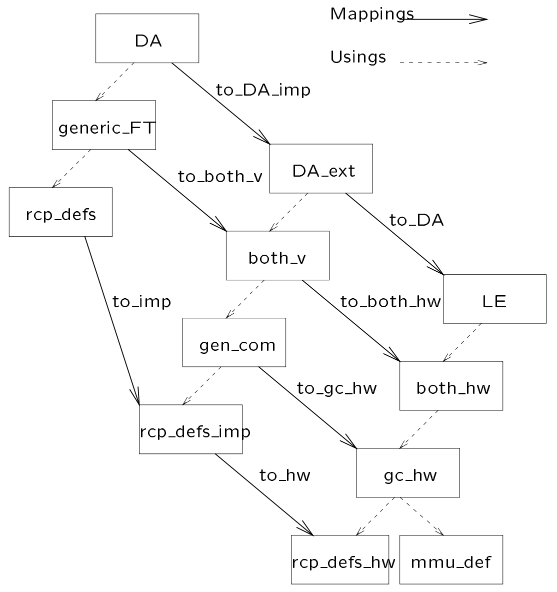
Illustration of Limitations



Fault Masking and Transient Recovery Through Exact-Match Voting

- Synchronous case developed by Rushby and formally verified in EHDM, 1991 (reveals need for interactive consistency)
- Mapping to more realistic loosely synchronized case (i.e., connection to clock synchronization) performed by Rick Butler and Ben DiVito (NASA), 1992
- Four Level hierarchy:
  - ◦ Uniprocessor synchronous
  - ◦ Replicated synchronous
  - ◦ Distributed synchronous
  - ◦ Distributed loosely synchronous
- Subsequently pushed down to hardware details (reveals need for address-space isolation)

Lowest Level Mappings



Current Work

- Working with Collins to verify microcode of next-generation commercial avionics processor (stack architecture, deep pipeline, prefetch)
- Working with IBM/Loral, JPL, NASA on requirements specification of "jet-select" function of Space Shuttle on-orbit DFCS in anticipation of major change in OI-24
- Working with Allied Signal on verification of MAFT algorithms for diagnosis, fault-tolerant scheduling
- (Soon) working with Allied, Boeing, Collins, FAA, Honeywell, NASA on fault partitioning for avionics (e.g., Safebus for 777)

Overview

- Examples of problems in critical systems: redundancy management in aircraft flight control systems
- What might help: formal methods
- Some examples of formal methods applied to redundancy management
- $\boxed{\text{Assurance for critical systems}}$

Modifications and Extensions to ICA Verification

- Erwin Liu (SRI) designed and formally verified hardware circuit needed to obtain accurate estimates of clock differences required by ICA
- Assumption that initial clock adjustments are zero found to be inconvenient
- Eliminated from specification, proofs re-run, adjustments made
- Dan Palumbo (NASA) experimented with different hardware implementation, found achieved synchronization tighter than predicted, and suggested adjustment to analysis
- Modified verification to accommodate this adjustment, and to extend analysis to a hybrid fault model in just a couple of days

More Clock Synchronization Verifications

- Almost all clock synchronization algorithms can be regarded as variations on a general paradigm due to Schneider
- General case, and instantiation for ICA formally verified by Shankar using EHDM, 1991
- As before, found numerous small flaws in original
- And developed streamlined argument
- Paul Miner (NASA) subsequently verified instantiation for the Lundelius-Lynch algorithm, and extended the analysis to include transient recovery

Interactive Consistency Algorithms

- Oral Messages Algorithm (OM): original Byzantine fault-tolerant algorithm due to Pease, Shostak, and Lamport (JACM, Apr 80; TOPLAS, Jul 82)
- Formally verified (together with an implementation) by Bevier and Young (CLI) using Boyer-Moore prover, 1990
- Algorithm verification later duplicated (and considerably simplified) by Rushby using EHDM
  (proof is an order of magnitude easier than ICA)

Interactive Consistency Algorithms

- Extension of OM to hybrid fault model proposed by Thambidurai and Park (SRDS, 1988)
- Formal verification attempted by Rushby, revealed bug in the algorithm
- Corrected algorithm for hybrid case developed by Lincoln (SRI) and Rushby using PVS, 1992 (much harder than pure Byzantine case because of additional case-splits, more arithmetic)

Lemmas (continued)

Lemma 3 *If the clock synchronization conditions S1 and S2 hold for i, channels p and q are nonfaulty through period $i+1$, and $T \in S^{(i)}$, then*

$$|c_p^{(i)}(T + \Delta_{q\,p}^{(i)}) - c_q^{(i)}(T)| < \epsilon + \rho S.$$

Lemma 4 *If the clock synchronization conditions S1 and S2 hold for i, channels p, q, and r are nonfaulty through period $i+1$, and $T \in S^{(i)}$, then*

$$|c_p^{(i)}(T) + \bar{\Delta}_{r\,p}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{r\,q}^{(i)}]| < 2(\epsilon + \rho S) + \rho\Delta.$$

Lemma 5 *If the clock synchronization condition S1 holds for i, channels p and q are nonfaulty through period $i+1$, and $T \in S^{(i)}$, then*

$$|c_p^{(i)}(T) + \bar{\Delta}_{r\,p}^{(i)} - [c_q^{(i)}(T) + \bar{\Delta}_{r\,q}^{(i)}]| < \delta + 2\Delta.$$

Formal Verification: Example

```
lemma3: **Module**

**Using** algorithm, lemma2

**Theory**

  $p, q$: **Var** proc
  $i$: **Var** period
  $T, T_0, T_1, T_2$: **Var** clocktime
  $\Pi$: **Var** realtime
  lemma3def: **Lemma**
    S1C$(p, q, i) \wedge S_2(p, i)$
        $\wedge$nonfaulty$(p, i+1) \wedge$ nonfaulty$(q, i+1) \wedge T \in S^{(i)}$
      $\supset |c_p^{(i)}(T + \Delta_{q\,p}^{(i)}) - c_q^{(i)}(T)| < \epsilon + \rho * S$
```

Example (continued)

```
lemma3_proof: **Prove** lemma3def **from**
  A2,
  rearrange_alt
    $\{x \leftarrow c_p^{(i)}(T + \Delta_{q\,p}^{(i)}),$
     $y \leftarrow c_q^{(i)}(T),$
     $u \leftarrow c_p^{(i)}(T_0@p1 + \Delta_{q\,p}^{(i)}),$
     $v \leftarrow T - T_0@p1,$
     $w \leftarrow c_q^{(i)}(T_0@p1)\},$
  lemma2b
    $\{T \leftarrow T_0@p1,\ \Phi \leftarrow \Delta_{q\,p}^{(i)},\ \Pi \leftarrow T - T_0@p1\},$
  lemma2c $\{p \leftarrow q,\ T \leftarrow T_0@p1,\ \Pi \leftarrow T - T_0@p1\},$
  rho_pos, nonfx, nonfx $\{p \leftarrow q\},$
  mult_mon2 $\{x \leftarrow |T - T_0@p1|,\ y \leftarrow S,\ z \leftarrow \frac{\rho}{2}\},$
  in_S_lemma $\{T_1 \leftarrow T,\ T_2 \leftarrow T_0@p1\}$

**End** lemma3
```

Details of Formal Verification for ICA

- About 3 man months of effort (at the time, one of the hardest computer-science verifications undertaken)
  - Could now be done in a couple of weeks

  24 modules (need supporting theories for induction, absolute value, summation, arithmetic mean, etc.)

- 1,300 lines of specification (formulas shown earlier taken directly from the formal specification)

- 19 axioms, 25 definitions, 182 proofs

- 10 minutes elapsed to check all proofs on a SparcStation 2

- Since duplicated by Bill Young (CLI) using Boyer-Moore prover

Examples: Synchronous Fault-Tolerant Systems

- Synchronous systems run the channels in lock-step, perform identical calculations in all channels, and use exact-match voting
- Overwhelmingly preferred by researchers in fault-tolerant systems since behavior is predictable
- Used in at least one DFCS developed by commercial manufacturer (MAFT by Allied Signal)
- Our interest: use of formal methods to develop and analyze algorithms and architectures for synchronous DFCS
- Verification should establish theorems of form
  if not too many/too bad faults **then** things OK
- Reliability modeling should calculate
  Prob[not too many/too bad faults]

Synchronous Fault-Tolerant Systems

- Must keep clocks synchronized so that all channels do the same thing at about the same time
- Need fault-tolerant distribution of sensor values so that each channel works on the same data—"interactive consistency" (aka. "source congruence" or "Byzantine agreement")
- Vote actuator outputs to mask channel faults
- And vote intermediate values for transient recovery
- Need to prove individual elements work, and that it all hangs together
- Must have an explicit fault model: types and stochastic properties of hardware faults that are to be tolerated

Clock Synchronization Algorithms and Implementations

- Interactive Convergence Algorithm (ICA): Byzantine fault-tolerant algorithm due to Lamport and Melliar-Smith (JACM, Jan 85)
- Formally verified by Rushby and von Henke using EHDM, 1988
- Verification carried down to very elementary axiomatic basis (Noetherian induction, function extensionality, and monotonicity of multiplication)
- Found proof of main theorem, and all but one of the lemmas are flawed in the original: e.g., main induction has the form
$$|c_p^{(i)}(T) - c_q^{(i)}(T)| < \delta \text{ implies } |c_p^{(i+1)}(T') - c_q^{(i+1)}(T')| \lesssim \delta$$
- Formal verification led to considerable simplification of the argument (and elimination of approximations)

Lemmas

The proof that A1, A2, A3, and C0 through C6 are sufficient to ensure that ICA achieves S1 and S2 depends on the following 5 lemmas.

Lemma 1 *If the clock synchronization conditions S1 and S2 hold for i, and channels p and q are nonfaulty through period $i+1$, then*
$$|\Delta_{q\,p}^{(i)}| < \Delta.$$

Lemma 2 *If channel p is nonfaulty through period $i+1$, and T and Π are such that $T + C_p^{(i)}$ and $T + \Pi + C_p^{(i)}$ are both in the interval $[T^{(0)} + C_p^{(0)}, T^{(i+2)} + C_p^{(i+1)}]$, then*
$$|c_p^{(i)}(T + \Pi) - [c_p^{(i)}(T) + \Pi]| \leq \frac{\rho}{2}|\Pi|.$$

Formal Methods

- Use of techniques from logic and discrete mathematics to model the requirements, specification, design, and implementation of computer systems
- In a way that supports *analysis* of certain properties (e.g., consistency, completeness)
- And *prediction* of (modeled) behavior
- Through systematic processes that resemble calculation

What Makes a Method Formal?

- A *formal* method is equipped with some *rules of deduction*
- These make it possible to *calculate* whether certain conclusions follow from certain premises
- The calculation is called a *proof*
- Because it is a calculation, it can be checked by others
- Or by a machine
- You don't have to understand what the symbols *mean* in order to check the calculation, you just have to know the rules for manipulating them (just like arithmetic calculations)

Proof and Truth

- Logic provides rules of calculation (i.e., of proof) that enable valid conclusions to be deduced from assumed premises
- Assumptions about the world are made *explicit* in the premises, separated from rules of deduction
- If the premises are true statements about the world
- Then the soundness theorems of logic guarantee that the conclusion is also a true statement about the world
- Need to *validate* the premises and the interpretation of the conclusion

Overview

- Examples of problems in critical systems: redundancy management in aircraft flight control systems
- What might help: formal methods
- Examples of formal methods applied to redundancy management
- Assurance for critical systems

Analysis: Dale Mackall, NASA Engineer AFTI F16 Flight Test

- Nearly all failure indications were not due to actual hardware failures, but to design oversights concerning asynchronous computer operation
- Failures due to lack of understanding of interactions among
  - Air data system
  - Redundancy management software
  - Flight control laws (decision points, thumps, ramp-in/out)

Overview

- Examples of problems in critical systems: redundancy management in aircraft flight control systems
- $\boxed{\text{What might help: formal methods}}$
- Some examples of formal methods applied to redundancy management
- Assurance for critical systems

An Interpretation of Mackall's Analysis

- Dealing with distributed, concurrent, real-time execution in the presence of faults
- Space of possible behaviors is vast
- Testing and simulation visit only small fraction of that space, and extrapolation from tested to untested cases is dubious (behavior is not continuous)
- Need a way to predict behavior under *all* circumstances
- (And a rational design to give some structure to the space)

Mathematical Modeling

- One way to predict behavior of a system is to construct a mathematical model and *calculate* it
- Model must be reasonably accurate
- And calculation must be performed without error
- For continuous systems, use well-developed mathematical theories (e.g., Navier-Stokes equations for aerodynamics)
- For computer systems, must use discrete mathematics (logic, set theory) and build our own theories
- And proofs of theorems take the place of numerical calculation

AFTI F16 Flight Test, Flight 44

- Asynchronous operation, skew, and sensor noise led each channel to declare the others failed
- Analog backup not selected (simultaneous failure of two channels not anticipated)
- Aircraft flown home on a single digital channel (not designed for this)
- No hardware failures had occurred

Other AFTI F16 Flight Tests

- Repeated channel failure indication in flight was traced to roll-axis software switch
- Sensor noise and asynchronous operation caused one channel to take a different path through the control laws
- Decided to vote the software switch
- Extensive simulation and testing performed
- Next flight, same problem still there
- Found that although switch value was voted, the unvoted value was used

X29A Flight Test

- Three sources of air data on X29A: nose and two side probes
- If value from nose is within threshold of both side probes, use nose probe value
- Threshold is large due to position errors in certain flight modes
- If nose probe failed to zero at low speed it would still be within threshold of correct readings
- Aircraft would become unstable and “depart”
- Caught in simulation but 162 flights had been at risk

HiMAT Flight Test

- Single failure in redundant uplink hardware
- Software detected this, and continued operation
- But would not allow the landing skids to be deployed
- Aircraft landed with skid retracted, sustained some damage
- Traced to timing change in the software that had survived extensive testing, but not in presence of failures

Redundancy Management in DFCS

- Hardware components of DFCS are far less reliable than the system requirement
- Hence, redundancy among sensors, actuators, and computing channels
- Impact of redundancy management is considerable
  - Typically more than 50% of code
  - Intrinsically difficult problems: coordination of distributed systems in presence of faults
  - Can become *primary* source of *un*reliability, and source of single point failure

Historical Experience With Asynchronous Designs

- Advanced Fighter Technology Integration (AFTI) F16
- Digital Flight Control System (DFCS) to investigate "decoupled" control modes
- Triplex DFCS to provide two-fail operative design
- Analog backup
- Digital computers not synchronized
- "General Dynamics believed synchronization would introduce a single-point failure caused by EMI and lightning effects"

AFTI F16 Flight Test, Flight 15

- Stores Management System (SMS) relays pilot requests for mode changes to DFCS
- An unknown failure in the SMS caused it to request mode changes 50 times a second
- DFCS responded at a rate of 5 mode changes per second
- Pilot said aircraft felt like it was in turbulence
- Analysis showed that if aircraft had been maneuvering at the time, DFCS would have failed

AFTI F16 Flight Test, Flight 36

- Control law problem led to "departure" of three seconds duration
- Sideslip exceeded $20°$, normal acceleration exceeded -4g, then $+7$g, angle of attack went to $-10°$, then $+20°$, aircraft rolled $360°$, vertical tail exceeded design load, failure indications from canard hydraulics, and air data sensor
- Side air data probe blanked by canard at high AOA
- Wide threshold passed error, different channels took different paths through control laws
- Analysis showed this would cause complete failure of DFCS and reversion to analog backup for several areas of flight envelope

Formal Methods: Instruments of Justification or Tools for Discovery?

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Overview

- Examples of problems in critical systems: redundancy management in aircraft flight control systems
- What might help: formal methods
- Some examples of formal methods applied to redundancy management
- Assurance for critical systems

Problem Domain

- Digital Flight Control Systems (DFCS)
- Computer systems that interpret pilot commands, sample sensors, evaluate control laws, and command the actuators
- Used on all modern warplanes, increasingly on commercial airplanes (Airbus A320, Boeing 777)
- Aircraft certification requires assurance that catastrophic failures will be "extremely improbable"
  - Not expected to occur in lifetime of fleet
  - "As an aid to engineering judgment" probability of catastrophic failure should be less than 10^{-9}/hour over a ten hour flight
- Many DFCS failures would be catastrophic

What Goes Wrong?

- Software for aircraft is developed to extremely rigorous standards, and subjected to massive testing
- Evidence is that design and coding bugs in sequential components are eliminated effectively
- Concern centers on
  - Requirements (e.g., JPL data for Voyager and Galileo spacecraft: only 3 of 197 mission-critical defects were programming problems; IBM data for Space Shuttle: 400 "user notes" documenting requirements anomalies—cf. 1 or 2 implementation defects)
  - The intrinsically hard problems (coordination of distributed computations, timing, synchronization, fault tolerance)

    These mainly arise in redundancy management